CLAIMS

We claim:

1. A computer-implemented method for executing a Java application on a client using data hosted on an EJB server, the method comprising:

generating an object graph descriptor;

creating a light-weight object graph, based on the object graph descriptor, the light-weight object graph being a subset of a domain object graph on the EJB server;

transporting the light-weight object graph to the client;

executing the Java application on the client;

monitoring a plurality of method invocations performed on the client that interact with the light-weight object graph;

logging a subset of method invocations of the plurality of method invocations that result in a change to the light-weight object graph in a stack; and

persisting changes to the domain object graph on the EJB server.

2. The method of claim 1 wherein transporting of the light-weight object graph includes serializing the light-weight object graph on the EJB server into a byte-stream, communicating the byte-stream to the client, and deserializing the byte-stream on the client.

3. The method of claim 1 wherein the object graph descriptor identifies at least one root object and at least one degree of traversal from the root object.

4. The method of claim 1 wherein the Java application is a Java graphical user interface application.

5. The method of claim 1 wherein the monitoring step is performed using a dynamic proxy.

6. The method of claim 5 wherein the monitoring step includes the dynamic proxy receiving a method call, the proxy executing the method call on an object in the light-weight object graph, and the proxy determining if the object changed as a result of the method call.

7. The method of claim 1 wherein persisting the changes to the domain object graph on the EJB server includes communicating the stack to the EJB server and executing the method calls thereon.

8. The method of claim 1 further including placing at least one flag in the stack and generating a corresponding snapshot of the light-weight object graph, prior to operation of the method invocation immediately following the flag, such that a user could cause the state of each of a plurality of objects in the light-weight object graph to return to a previous state.

9. The method of claim 1 wherein executing a Java application on the client includes generating a container aware model adapted to allow the Java application to retrieve data from the light-weight objects embedded in the light-weight object graph.

10. The method of claim 9 wherein executing a Java application on the client includes creating a form in communication with the light-weight object graph through the container aware model.

11. The method of claim 10 wherein the container aware model is created by extending a standard Java Swing model.

12. The method of claim 3 wherein creating a light-weight object graph includes:

locating the root entity bean specified by the at least one root object in the domain object graph;

locating at least one related entity bean in the domain object graph, as specified by the degree of traversal; and

returning a first implementation of the light-weight class from which the root entity bean inherits and a second implementation of the light-weight class from which the related entity bean inherits.

13. The method of claim 12 wherein the first implementation and the second implementation are generated by an object provider.

14. The method of claim 1 further including extending the light-weight object graph, based on an extension request from the client identifying an extension root and an extension degree of traversal, creating a light-weight version of the corresponding objects, and transporting the corresponding objects to the client.

15. The method of claim 1 wherein the domain object graph on the EJB server includes a collection of heavy-weight entity beans and the entity beans are created from:

a foundational light-weight object class specifying a set of states and a set of behaviors associated with a set of light-weight objects;

at least one specific light-weight object class specifying a second set of states and set of behaviors associated with the specific light-weight object class, the specific light-weight object class inheriting from the foundational light-weight object class; and

at least one instance of the at least one specific light-weight object class, the at least one instance including information corresponding to the first and second set of states.

16. The method of claim 4 wherein the Java graphical user interface application is a Java Swing application.

17. The method of claim 16 wherein the Java Swing application is a container-aware Java Swing application.

18. A computer program product residing on a computer-readable medium for use in creating a client container for interacting with an EJB server, the computer program product comprising:

first instructions for creating a light-weight object graph, based on an object graph descriptor, the light-weight object graph being a subset of a domain object graph on the EJB server;

second instructions for transporting the light-weight object graph to the client;

third instructions for monitoring a plurality of method calls performed on the client that interact with the light-weight object graph;

fourth instructions for logging in a stack a subset of method calls of the plurality of method calls that result in a change to the light-weight object graph; and

fifth instructions for persisting changes to the domain object graph on the EJB server.

19. The method of claim 18 wherein the second instructions include instructions for serializing the light-weight object graph on the EJB server into a byte-stream, communicating the byte-stream to the client, and deserializing the byte-stream on the client.

20. The method of claim 18 wherein the object graph descriptor identifies at least one root object and at least one degree of traversal from the root object.

21. The method of claim 20 wherein the first instructions include instructions for:

locating the root entity bean specified by the at least one root object in the domain object graph;

locating at least one related entity bean in the domain object graph, as specified by the degree of traversal; and

returning a first implementation of the light-weight class from which the root entity bean inherits and a second implementation of the light-weight class from which the related entity bean inherits.

22. The method of claim 18 including sixth instructions for placing at least one flag in the stack and generating a corresponding snapshot of the light-weight object graph, prior to operation of the method invocation immediately following the flag, such that a user could cause the state of each of a plurality of objects in the light-weight object graph to return to a previous state.

23. The method of claim 18 wherein persisting the changes to the domain object graph on the EJB server includes communicating the stack to the EJB server and executing the method calls thereon.

24. The method of claim 23 wherein the fifth instruction include instruction for creating a session bean responsible for executing the method calls in the stack on the EJB server.

25. The method of claim 18 further including sixth instructions for extending the light-weight object graph, based on an extension request from the client identifying an extension root and an extension degree of traversal, creating a light-weight version of the corresponding objects, and transporting the corresponding objects to the client.

26. The method of claim 18 further including sixth instructions for defining a set of container-aware Java graphical user interface applications.

27. The method of claim 26 further including instructions for defining a container-aware model, as an extension of a standard Java Swing model, the container-aware model adapted to allow the Java application to retrieve data from a plurality of light-weight objects embedded in the light-weight object graph.

28. The method of claim 18 wherein the third instructions include instructions for using a dynamic proxy to receive a method call, execute the method call on an object in the light-weight object graph, and determine if the object changed as a result of the method call.

29. The method of claim 18 wherein the domain object graph on the EJB server includes a collection of heavy-weight entity beans and the entity beans are created from:

a foundational light-weight object class specifying a set of states and a set of behaviors associated with a set of light-weight objects;

at least one specific light-weight object class specifying a second set of states and set of behaviors associated with the specific light-weight object class, the specific light-weight object class inheriting from the foundational light-weight object class; and

at least one instance of the at least one specific light-weight object class, the at least one instance including information corresponding to the first and second set of states.

30. A computer system for executing a Java application on a client using data hosted on an EJB server, the system comprising:

a first storage medium located on the EJB server, the first storage medium including a domain object graph consisting of a plurality of heavy-weight objects;

a servlet located on the EJB server, the servlet configured to execute computer code adapted to:

(i)create a light-weight object graph, based on an object graph descriptor, the light-weight object graph being a subset of the domain object graph; and

(ii)transport the light-weight object graph to the client;

a processor located on the client; and

a second storage medium located on the client, the second storage medium storing the light-weight object graph and computer code adapted to:

(i)execute the Java application; and

(ii)log a subset of method calls of the plurality of method calls that result in a change to the light-weight object graph;

31. The method of claim 30 wherein the client computer code is further adapted to return the subset of method calls to the EJB server, and the servlet is further adapted to update the domain object graph, based on any change functions at the client that affected a state of a light-weight object within the light-weight object graph.

32. The method of claim 30 wherein the object graph descriptor identifies at least one root object and at least one degree of traversal from the root object.

33. The method of claim 30 wherein the instructions for creating a light-weight object graph include instructions for:

locating the root entity bean specified by the at least one root object in the domain object graph;

locating at least one related entity bean in the domain object graph, as specified by the degree of traversal; and

returning a first implementation of the light-weight class from which the root entity bean inherits and a second implementation of the light-weight class from which the related entity bean inherits.

34. The method of claim 30 wherein transporting of the light-weight object graph includes serializing the light-weight object graph on the EJB server into a byte-stream, communicating the byte-stream to the client, and deserializing the byte-stream on the client.

35. The method of claim 30 wherein the Java application is a Java graphical user interface application.

36. The method of claim 30 wherein the code adapted to log a subset of method calls includes code for using a dynamic proxy.

37. The method of claim 36 wherein the code adapted to log a subset of method calls includes code for using the dynamic proxy for receiving a method call, executing the method call on an object in the light-weight object graph, and determining if the object changed as a result of the method call.

38. The method of claim 30 wherein executing a Java application on the client includes generating a container aware model adapted to allow the Java application to retrieve data from the light-weight objects embedded in the light-weight object graph.

39. The method of claim 38 wherein executing a Java application on the client includes creating a form in communication with the light-weight object graph through the container aware model.

40. The method of claim 30 wherein the domain object graph on the EJB server includes a collection of heavy-weight entity beans and the entity beans are created from:

a foundational light-weight object class specifying a set of states and a set of behaviors associated with a set of light-weight objects;

at least one specific light-weight object class specifying a second set of states and set of behaviors associated with the specific light-weight object class, the specific light-weight object class inheriting from the foundational light-weight object class; and

at least one instance of the at least one specific light-weight object class, the at least one instance including information corresponding to the first and second set of states.

41. A computer-implemented method for executing an application on a client using a domain object graph hosted on an EJB server, the method comprising:

generating an object graph descriptor identifying at least one root object and at least one degree of traversal from the root object;

creating a light-weight object graph, based on the object graph descriptor, the light-weight object graph being a subset of the domain object graph on the EJB server; and

transporting the light-weight object graph to the client.

42. The method of claim 41 wherein transporting the light-weight object graph includes serializing the light-weight object graph on the EJB server into a byte-stream, communicating the byte-stream to the client, and deserializing the byte-stream on the client.

43. The method of claim 41 wherein creating a light-weight object graph includes:

locating the root entity bean specified by the at least one root object in the domain object graph;

locating at least one related entity bean in the domain object graph, as specified by the degree of traversal; and

returning a first implementation of the light-weight class from which the root entity bean inherits and a second implementation of the light-weight class from which the related entity bean inherits.

44. The method of claim 43 wherein the first implementation and the second implementation are generated by an object provider.

45. The method of claim 40 further including extending the light-weight object graph, based on an extension request from the client identifying an extension root and an extension degree of traversal, creating a light-weight version of the corresponding objects, and transporting the corresponding objects to the client.

46. The method of claim 40 wherein the domain object graph on the EJB server includes a collection of heavy-weight entity beans and the entity beans are created from:

a foundational light-weight object class specifying a set of states and a set of behaviors associated with a set of light-weight objects;

at least one specific light-weight object class specifying a second set of states and set of behaviors associated with the specific light-weight object class, the specific light-weight object class inheriting from the foundational light-weight object class; and

at least one instance of the at least one specific light-weight object class, the at least one instance including information corresponding to the first and second set of states.

47. A computer-implemented, object-oriented programming model for facilitating execution of an application on a client using data stored on an EJB server, the model including:

a foundational light-weight object class specifying a set of states and a set of behaviors associated with a set of light-weight objects;

at least one specific light-weight object class specifying a second set of states and set of behaviors associated with the specific light-weight object class, the specific light-weight object class inheriting from the foundational light-weight object class;

at least one instance of the at least one specific light-weight object class, the at least one instance including information corresponding to the first and second set of states; and

at least one entity bean including a third set of behaviors, the entity bean inheriting from the at least one instance.

48. A method of resolving changes between a light-weight object graph located on a client and a domain object graph located on an EJB server, the method comprising:

monitoring a plurality of method invocations performed on the client that interact with the light-weight object graph;

logging a subset of method invocations of the plurality of method invocations that result in a change to the light-weight object graph in a stack;

transporting the stack to the EJB server; and

executing the method calls on the domain object graph.

49. The method of claim 48 wherein transporting of the stack includes serializing the plurality of method invocations into a byte-stream, communicating the byte-stream to the EJB server, and deserializing the byte-stream on the EJB server.

50. The method of claim 48 wherein the method invocations are executed on the EJB server using a session bean.

51. The method of claim 48 wherein the monitoring step is performed using a dynamic proxy.

52. The method of claim 48 wherein the monitoring step includes the dynamic proxy receiving a method invocation, the proxy executing the method invocation on an object in the light-weight object graph, and the proxy determining if the object changed as a result of the method invocation.